Vector Quantisation

Subject:	A Technology Discussion
Author:	VideoLogic Ltd (PowerVR)
File Name:	Vector Quantisation.doc
Issue Number:	1.0.8
Issue Date:	26 October 98
Copyright:	© VideoLogic Ltd 1998. All rights reserved. This material may not be republished in any format without the express consent of its authors. Information contained herein is provided "as-is", without representations or warranties, and is subject to change without notice.

Note

This is part of a paper written by Mark Butler and presented by Marc Pinter-Krainer at the Computer Games Developers Conference 1998.

Author/Speaker

Mark Butler - Programme Manager, VideoLogic Ltd. <mark@videologic.com>

Marc Pinter-Krainer – Developer Relations, VideoLogic Ltd. <mpk@videologic.com>

Abstract

In depth discussion on a specific feature: Vector Quantisation texture compression (providing up to 8:1 compression)

Target Audience (Intermediate Track: Intermediate Level)

3D artists and programmers interested in learning some of the latest 3D graphics techniques.

Audience prerequisites

General knowledge on the issues associated with texture compression. Previous experience with PowerVR is an asset but not a must.

Introduction

The use of a texture compression system allows a game to utilise many more and much larger textures. It is important that the method does not have a performance impact on the rendering and that it will compress typical texture images well. Vector Quantisation (VQ) has these properties. It has a high compression ratio and looks very good on surface textures and sharp edged detail like text. The method will be discussed from the user's point of view as well as the technical details.

Vector Quantisation is a method of compression that finds similar sections or blocks of an image, creates a table of the blocks and then encodes the image using indices into that code book table. It performs well on high frequency images like text and also on texture surface images.

User View

You do not have to get artists to rework your artwork to use VQ. The compression can be used on all your existing textures. Start by compressing all your textures with VQ. If this gives acceptable results then your work is done. If some of the compressed textures are not of high enough quality, then compress the rest and leave these uncompressed.

Support for VQ can be added in several places:

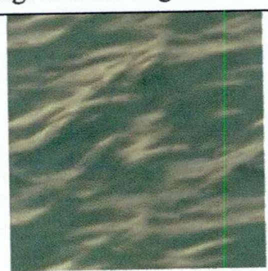
- External application creating VQ files
- Plug in to the image handlers/collateral compilers
- Library code added to application
- Driver functions

The amount of compression depends on the size of the image and the code book table size. For example a 256x256, 16 bit texture compressed using VQ 2x2 grid 8 bit index achieves a 7.1 to 1 compression ratio. See the section below for a full description.

VQ normally provides excellent quality for the type of textures used in games. This and the high compression ratios make it very attractive to the 3D games industry.

Comparisons

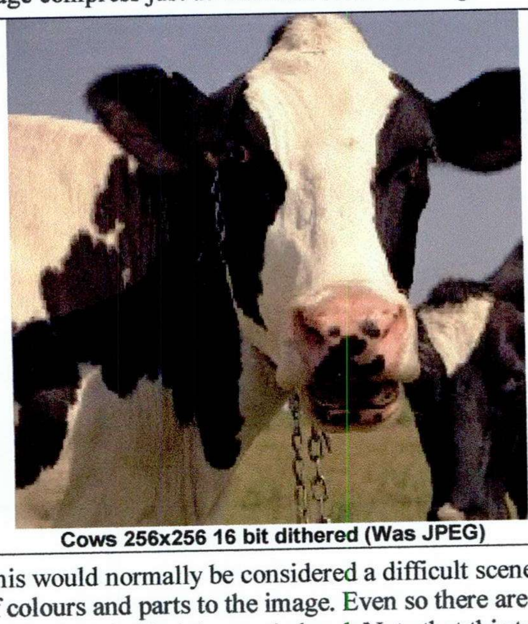
The following images have all been compressed with VQ using 2x2 blocks and a 256-entry table. When studying the images for compression artefacts, remember that these are the flat images and have not been used in a rendered scene. As part of the scene they will typically have been filtered, smooth shaded, fogged and so on. Each operation in the texturing and shading pipeline will help to obscure any differences between the compressed image and the original.



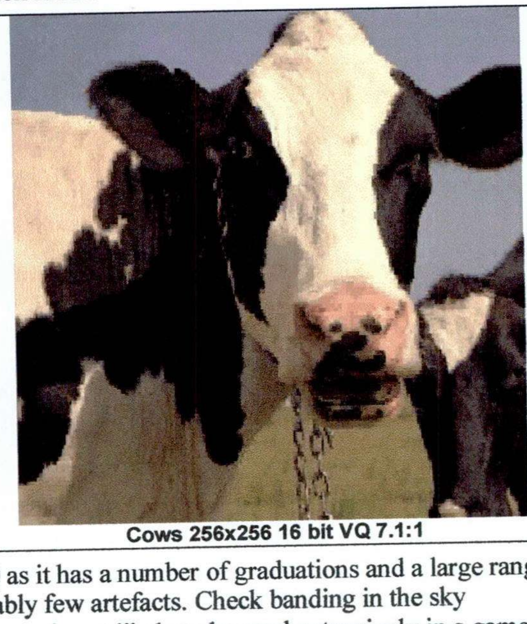
Water 128x128 24 bit

Water 128x128 24 bit VQ 6.9:1

A typical image used as a texture. This compresses very well with negligible differences. Larger versions of this image compress just as well and offer even higher compression ratios.

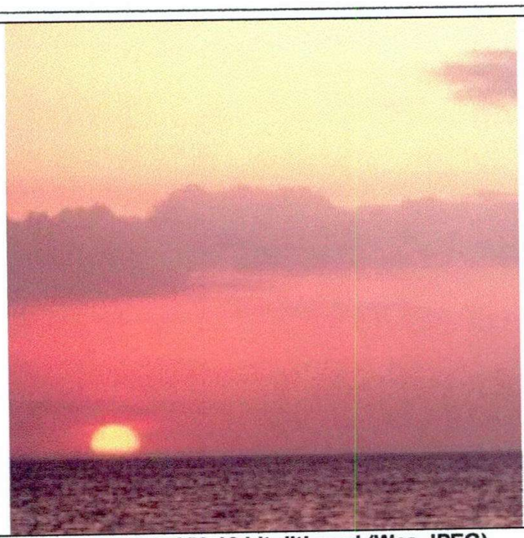


Cows 256x256 16 bit dithered (Was JPEG)

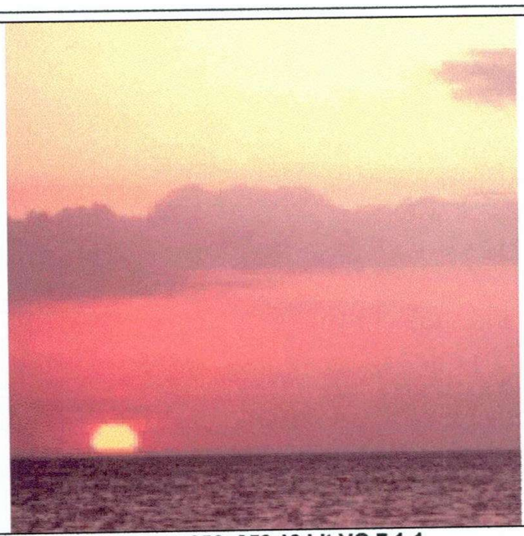


Cows 256x256 16 bit VQ 7.1:1

This would normally be considered a difficult scene for VQ as it has a number of graduations and a large range of colours and parts to the image. Even so there are remarkably few artefacts. Check banding in the sky graduations behind the cow's head. Note that this type of image is not likely to be used extensively in a game.



Sunset 256x256 16 bit dithered (Was JPEG)



Sunset 256x256 16 bit VQ 7.1:1

This is potentially a difficult scene for VQ due to the graduations. However it compresses well because there are few separate colour ranges. This allows the table to use more entries to cover the graduation.

Forest 256x256 16 bit dithered (Was JPEG)

Forest 256x256 16 bit VQ 7.1:1

A good example case for VQ. The image has detail and sharp edges, there is even a fair colour range and yet the compressed image is almost indistinguishable. However, careful checking can detect the drop in the yellow components of the centre leaf. This can be addressed by re-compressing with different parameters.

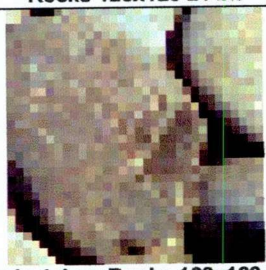
Technology View

This worked example will be done with a 128x128, 24 bit image. The original image is shown below with the resulting VQ image for comparison. The example uses a grid of 2x2 pixels. Each 2x2 block is compared against the other 2x2 blocks to find similar copies.

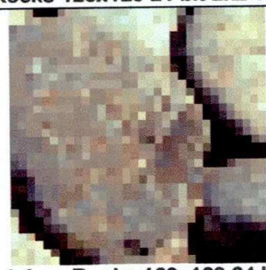
The search is similar to a palette selection operation. The constraint is the fixed number of entries available in the table, so the objective is to find a set of values that reduce the error for the image.

Rocks 128x128 24 bit

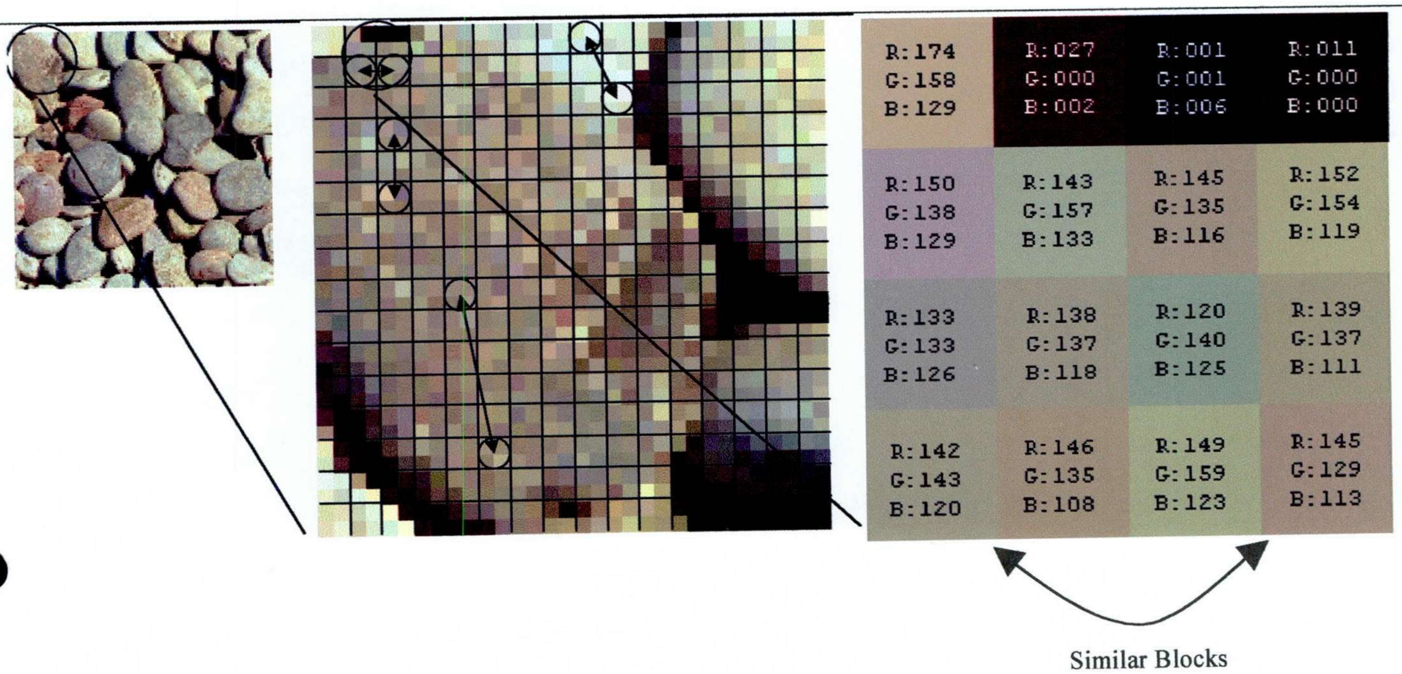
Rocks 128x128 24 bit 2x2 VQ



Zoom in 4:1 on Rocks 128x128 24 bit

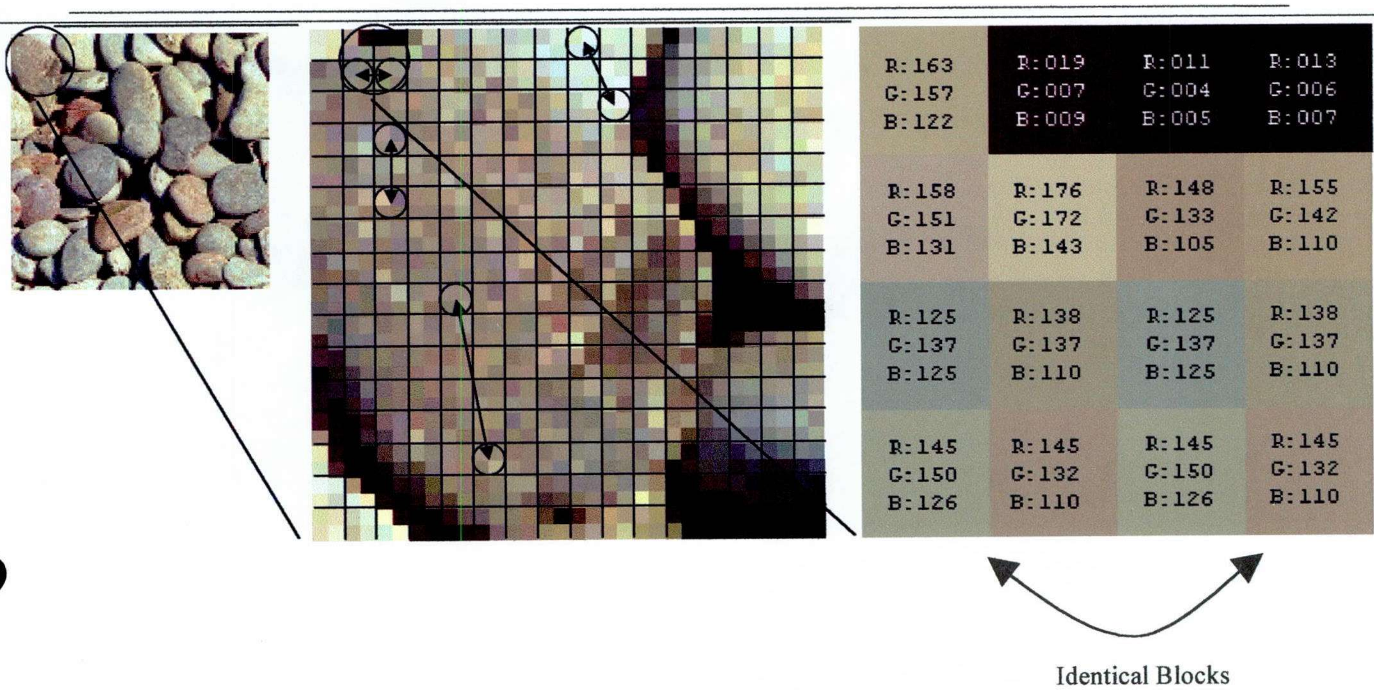


Zoom in 4:1 on Rocks 128x128 24 bit 2x2 VQ



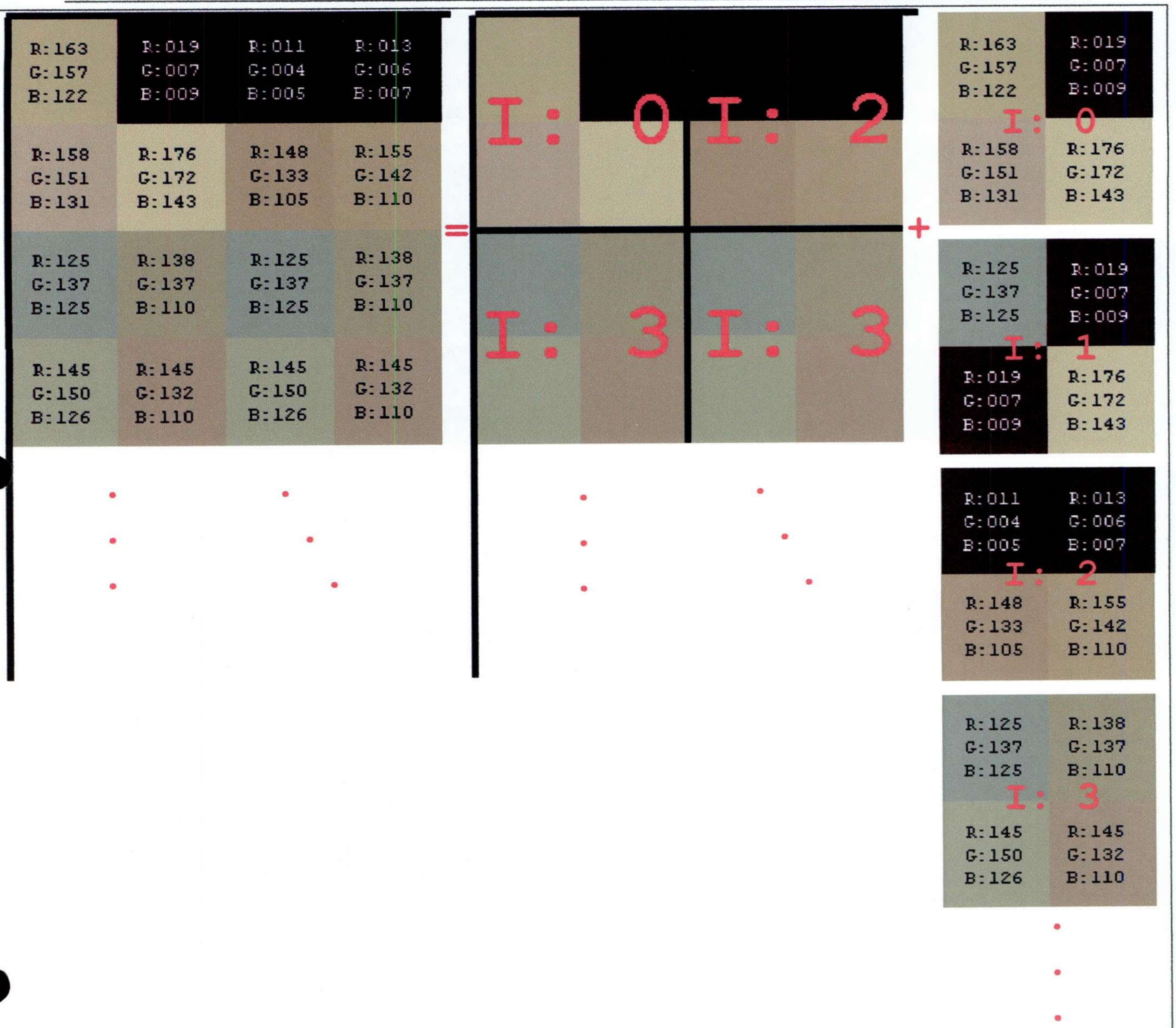
Original image is processed for similar pixel quads on a 2x2 block

The original image's data is stored in the RGB format, where each pixel contains 24 bit data. 8 bit for each red, green and blue component.



Modified image with duplicate 2x2 pixel blocks

During the selection process the 2x2 block will be changed slightly so that several blocks that do not match exactly will match with very little error. Now there are duplicates the data can be stored in a much more compact format.



The image is now stored as a series of indices (Index Map) plus the look up table.

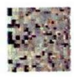
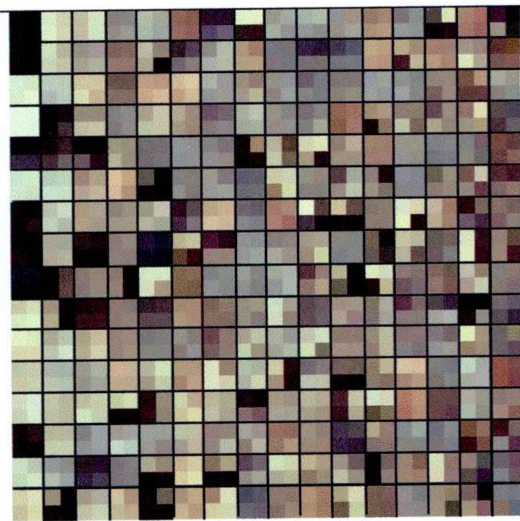


Image of the Lookup Table of 2x2 Codes



Expanded Lookup Table

Now the compression can be calculated. The original image is 128x128 and now consists of 64x64 index entries, each 8 bit. This makes 64x64= 4096 bytes, The table is 256 entries each consisting of 4 RGB entries, each entry is 24 bit data or 3 bytes. This makes 256x4x3= 3042. The total for the VQ image is 7138 bytes. The original image is 128x128 RGB entries each of 3 bytes. This makes 128x128x3= 49152 bytes: a compression ratio of 6.9 to 1.

Compression Ratios

The worked example above made a number of choices. The follow is a list of the items that need consideration when using VQ. They all affect the compression ratios and quality of the output:

- Block Size (E.G. 2x2 or 3x3)
- Source Bit Depth (E.G. 24 bit or 16 bit)
- Target Bit Depth (E.G. 24 bit or 16 bit)
- Number of Table Entries (E.G. 256 or 64) or
  Size of the Index values (E.G. 8 bit or 6 bit)

The worked example used 24 bit to simplify the discussion, however in a hardware implementation it is likely that 16 bit is a more natural choice.

A set of selections has been made in each of the tables below. They show the effect on the compression ratio at different image resolutions. In general the formula to work out the number of bytes in the VQ image is as follows:

VQ Image Size = IndexMapSize + TableMapSize
= ResX/BlockSizeX*ResY/BlockSizeY*VQBits/8 +
2^VQBits*BlockSizeX*BlockSizeY*BitDepth/8

The following example is a nice balance between the option to generate a good compression and a good image quality. This is the set of options used by default in PowerVR Second Generation.

Compression for 16 bit, 2x2, 8 bit VQ
[Res/2*Res/2*8Bit/8 + 256*2x2*16bit/8]

2*8Bit/8 + 256*2x2*16b	Image Bytes	VQ Image Bytes	Compression Ratio
64	8,192	3,072	2.7
128	32,768	6,144	5.3
256	131,072	18,432	7.1
512	524,288	67,584	7.8
1,024	2,097,152	264,192	7.9

Note, however, that the compression ratio for small images is not very good. This is because the code book table is larger than it needs to be. To address this the number of bits used for the indices can be reduced, this leads to a smaller code book table. For 6 bit VQ the table has 64 entries. Variable table size is an option on PowerVR Second Generation.

Compression for 16 bit, 2x2, 6 bit VQ
[Res/2*Res/2*6Bit/8 + 64*2x2*16bit/8]

2*8Bit/8 + 256*2x2*16b	Image Bytes	VQ Image Bytes	Compression Ratio
64	8,192	1,280	6.4
128	32,768	3,584	9.1
256	131,072	12,800	10.2
512	524,288	49,664	10.6
1,024	2,097,152	197,120	10.6

For very large images the block size can be increased and the quality of the image is retained. This also leads to an increased compression ratio for the large images, but not for the smaller sizes.

Compression for 16 bit, 3x3, 8 bit VQ
[Res/3*Res/3*8Bit/8 + 256*3x3*16bit/8]

2*8Bit/8 + 256*2x2*16b	Image Bytes	VQ Image Bytes	Compression Ratio
64	8,192	5,063	1.6
128	32,768	6,428	5.1
256	131,072	11,890	11.0
512	524,288	33,735	15.5
1,024	2,097,152	121,116	17.3

Performance

When using VQ compressed textures a lookup table entry and an index is required for each texel fill. This is a similar operation to a palletised texture. It may be look as if it would take twice as long in the hardware pipeline, compared to a single 16 bit fetch. However, the table is much smaller than a full size texture is suitable for caching techniques. This is particularly true during bilinear operations when it is likely that the surrounding texels are also needed. Performance is also offset against the need to only fetch 8 bits for the index as opposed to 16 bits for a full texel value. Thus a VQ texture implementation can be designed to perform as well as a normal texture.

Summary

VQ is a compression that works well for 3D games. It has a high compression ratio and is easy to use. Even if some of your textures remain in normal uncompressed texture surfaces, a typical case of 4MB of on board texture memory allows around 20MB of textures to be uploaded.

